DevOps Capstone Project 2

You are hired as a Devops engineer for XYZ.pvt.co, The company is a product based company they are using Docker containers for their containerization inside the company, but in the meantime, the product got a lot of traffic, now they need to have a platform for automating deployment, scaling, and operations of application containers across clusters of hosts, As a Devops engineer, you need to work on this and implement a Devops life cycle, such that the Docker containers in the testing environment will not change.

As the company is a monolithic architecture with 2 developers and the product is present on https://github.com/hshar/website.git

Following are the specifications of life-cycle:

1. Git workflow should be implemented, as the company is a monolithic architecture you need to take care of versions. The release should happen only on 25 of every month.
2. Code build should be triggered once the commits are made in the master or hotfix branch.
3. The code should be containerized with the help of the Docker file, The Dockerfile should be built every time if there is a push to git-hub. Use the container with Ubuntu and apache installed in it. After the build, this container should be pushed to the Docker hub.

4. The above tasks should be done in a Jenkins pipeline with the following jobs.
   1. Build website
   2. Test website
   3. Push to Docker hub
   4. Push to production

5. As per the requirement In the production server, you need to use the Kubernetes cluster and the containerized code from Docker hub should be deployed with 2 replicas. Use kubernetes dashboard for health checks of those containers using dashboard.
6. Once the application is built on the production server you need to design a test case, Which will basically check whether the configurations are displaying on the website or not. The test should pass if the configurations are displayed for the product on both the production and testing server.
7. For configuration management of the product, you need to deploy the configuration file in '/home/ubuntu' for the execution of the configuration in both testing and production server. The above task should be accomplished with the help of Ansible roles.
8. Create a monitoring service for the website on the production server. Before that, as a Devops engineer test this monitoring service by stopping the apache service and check whether the email is sent to you or not.

Architectural advice:

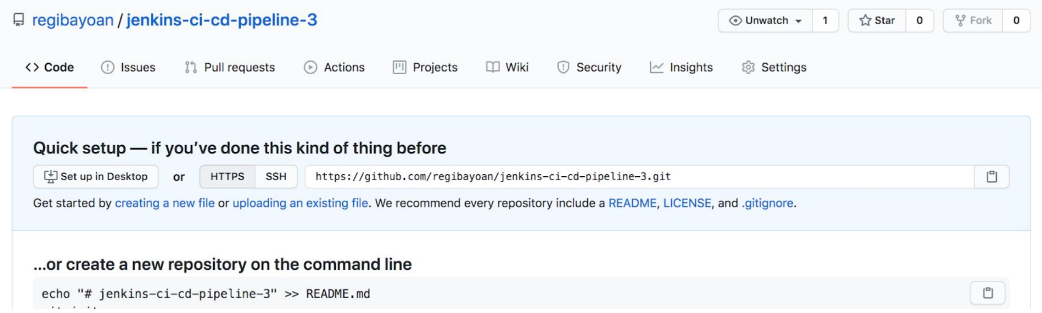
Server1 jenkins master,Nagios master
Server2 jankins slave,Testing server, nrpe plugin, PHP
Server3 jenkins slave, production server,nrpe plugin, kubernetes master, PHP
Server4 kubernetes node,nrpe plugin, host machine.

Solution

Step 1: Setup AWS architecture

- 4 servers => Three ”t2.micro”, One “t2.medium” (K8s Master Node requires 2 vCPUs)
- Server 1 => Jenkins master, Ansible master, Nagios master
- Server 2 => Testing server, Jenkins slave, nrpe plugin, Ansible host
- Server 3 => Production server, Jenkins slave, nrpe plugin, Ansible host, k8s master
- Server 4 => k8s worker node, nrpe plugin

Step 2: Create a new GItHub repository



regibayoan / jenkins-ci-cd-pipeline-3

Quick setup — if you’ve done this kind of thing before

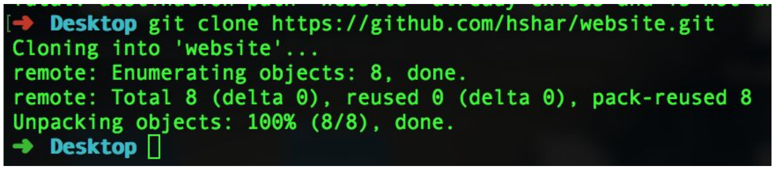
https://github.com/regibayoan/jenkins-ci-cd-pipeline-3.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

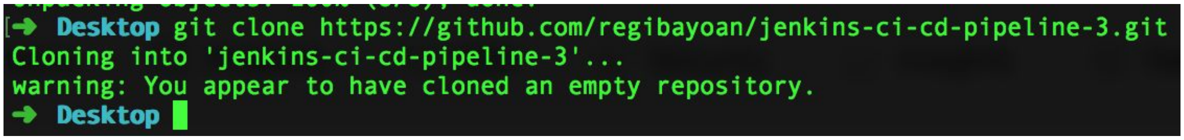
...or create a new repository on the command line

```
echo "# jenkins-ci-cd-pipeline-3" >> README.md
```

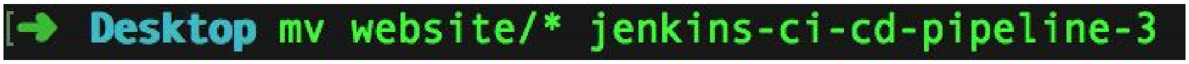
- Clone the website repository and move its contents to our new repository we just created
- Then remove the **index.html**. We will use an index.html that will be deployed by the Ansible server



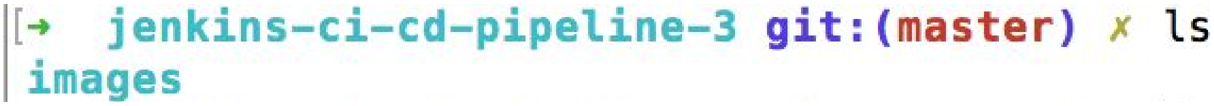
```
[➜ Desktop git clone https://github.com/hshar/website.git
Cloning into 'website'...
remote: Enumerating objects: 8, done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 8
Unpacking objects: 100% (8/8), done.
➜ Desktop
```



```
[➜ Desktop git clone https://github.com/regibayoan/jenkins-ci-cd-pipeline-3.git
Cloning into 'jenkins-ci-cd-pipeline-3'...
warning: You appear to have cloned an empty repository.
➜ Desktop
```

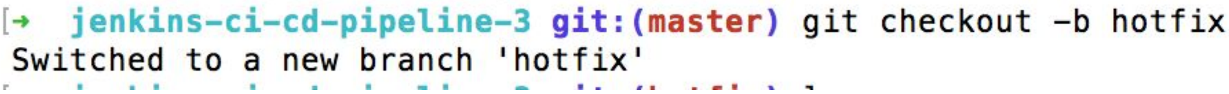


```
[➜ Desktop mv website/* jenkins-ci-cd-pipeline-3
```



```
[➜ jenkins-ci-cd-pipeline-3 git:(master) ✗ ls
images
```

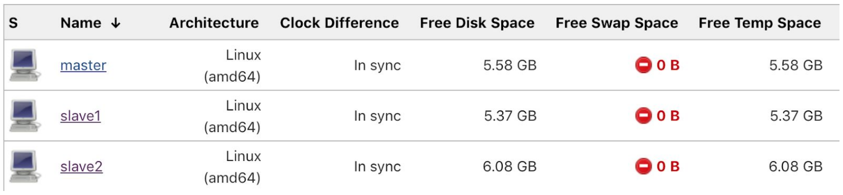
- Create a new branch -> **‘hotfix’**



```
[➜ jenkins-ci-cd-pipeline-3 git:(master) git checkout -b hotfix
Switched to a new branch 'hotfix'
```

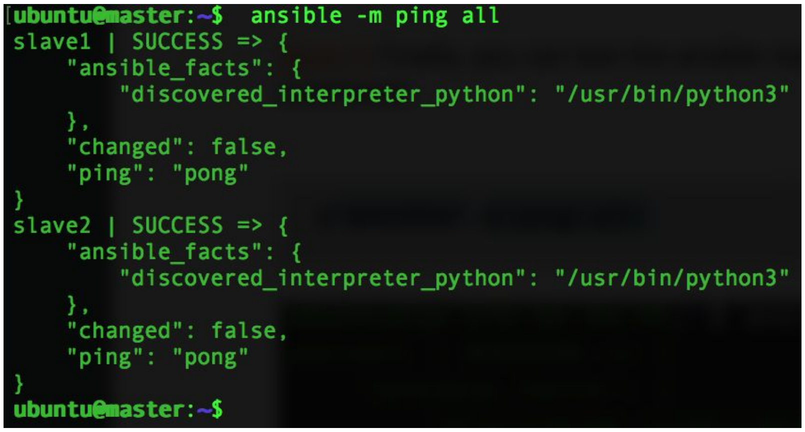
Step 3: Setup **Jenkins**, **Ansible** and **Nagios** on **Master** server

- Connect **Test** server and **Production** server to **Master** server as well
- After **Jenkins** setup it should look like this:



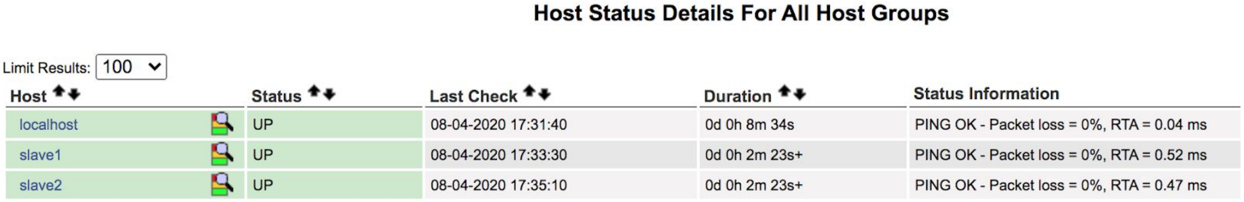
S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space
	master	Linux (amd64)	In sync	5.58 GB	⛔ 0 B	5.58 GB
	slave1	Linux (amd64)	In sync	5.37 GB	⛔ 0 B	5.37 GB
	slave2	Linux (amd64)	In sync	6.08 GB	⛔ 0 B	6.08 GB

- After **Ansible** setup we should be able to ping both servers :



```
ubuntu@master:~$  ansible -m ping all
slave1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
slave2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@master:~$
```

- After **Nagios** setup we should be able to monitor both servers:



Host Status Details For All Host Groups

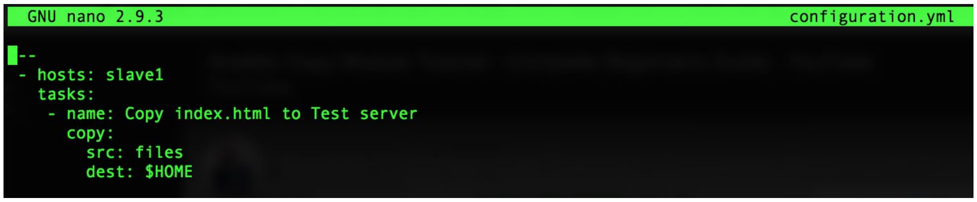
Limit Results: 100

Host	Status	Last Check	Duration	Status Information
localhost	UP	08-04-2020 17:31:40	0d 0h 8m 34s	PING OK - Packet loss = 0%, RTA = 0.04 ms
slave1	UP	08-04-2020 17:33:30	0d 0h 2m 23s+	PING OK - Packet loss = 0%, RTA = 0.52 ms
slave2	UP	08-04-2020 17:35:10	0d 0h 2m 23s+	PING OK - Packet loss = 0%, RTA = 0.47 ms

- Create a Nagios monitoring service using the guide below: https://docs.google.com/document/d/e/2PACX-1vRDn0sdlszCHfzVnMtO9WNkdqRBBMJpmtN2s2cKjAsJFI1ZacLVbY5pKHGPXWRY_1iik9-mjZFbNOgv/pub

Step 4: For configuration management, we will use Ansible roles to deploy the **index.html** to ***/home/ubuntu*** of the Test server. This will be used for building the Docker image.

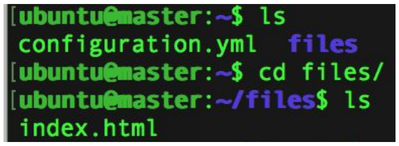
- First, create a playbook in Jenkins Master -> **configuration.yml**



```
  GNU nano 2.9.3                                              configuration.yml

---
- hosts: slave1
  tasks:
   - name: Copy index.html to Test server
     copy:
       src: files
       dest: $HOME
```

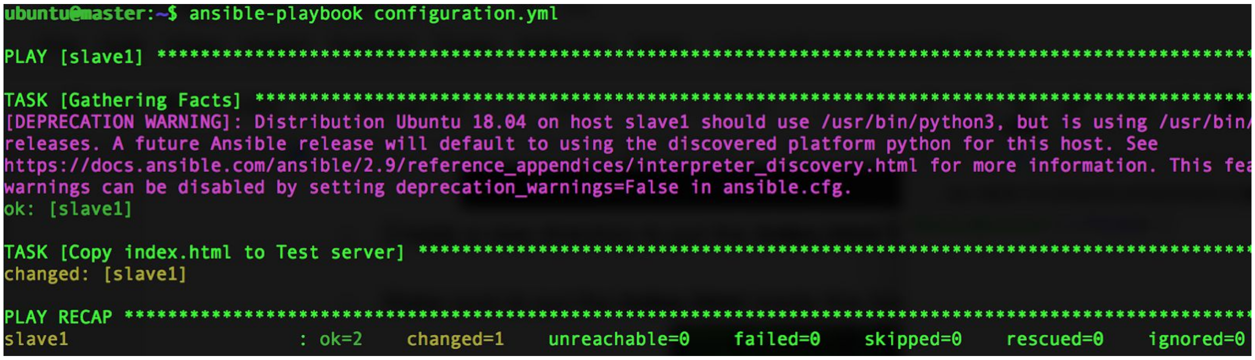
- Create a new directory to put the **index.html** file in -> **mkdir files**

- Make sure to put the **index.html** inside this folder

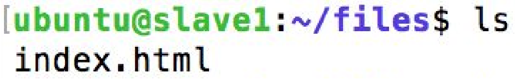


```
[ubuntu@master:~$ ls
 configuration.yml  files
[ubuntu@master:~$ cd files/
[ubuntu@master:~/files$ ls
 index.html
```

- Try running the playbook command to verify if we can copy the files to the Test server

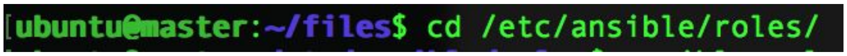


```
ubuntu@master:~$ ansible-playbook configuration.yml

PLAY [slave1] ********************************************************************************************

TASK [Gathering Facts] ***********************************************************************************
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host slave1 should use /usr/bin/python3, but is using /usr/bin/
releases. A future Ansible release will default to using the discovered platform python for this host. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more information. This fea
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
ok: [slave1]

TASK [Copy index.html to Test server] ********************************************************************
changed: [slave1]

PLAY RECAP ***********************************************************************************************
slave1                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```



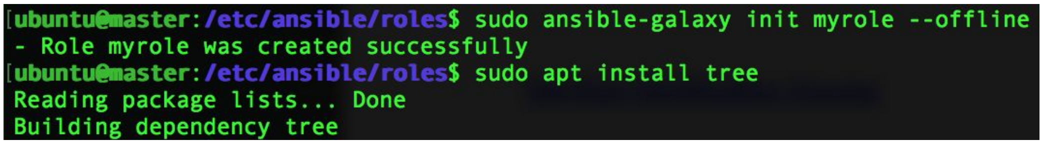
```
[ubuntu@slave1:~/files$ ls
 index.html
```

- Next, create Ansible roles
- First cd into /etc/ansible/roles

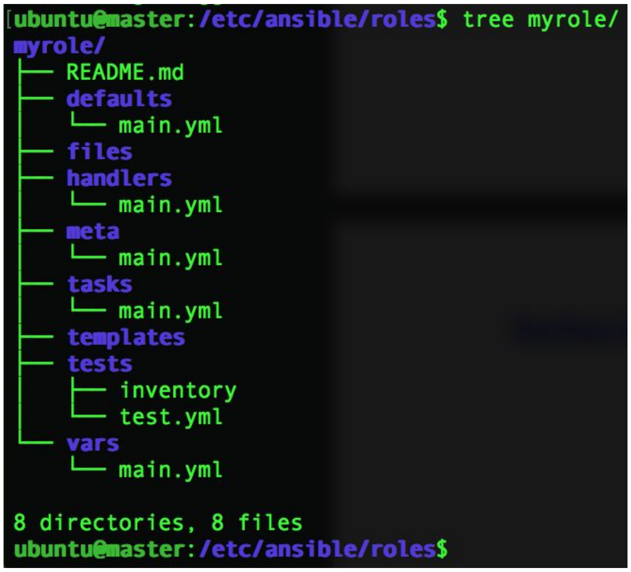


```
[ubuntu@master:~/files$ cd /etc/ansible/roles/
```

- Create the role and install **tree** to view our role in a tree structure

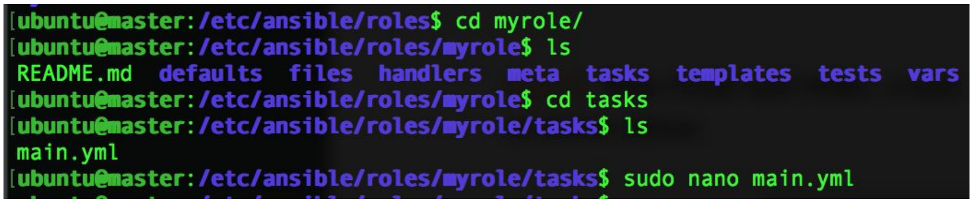


```
[ubuntu@master:/etc/ansible/roles$ sudo ansible-galaxy init myrole --offline
 - Role myrole was created successfully
[ubuntu@master:/etc/ansible/roles$ sudo apt install tree
 Reading package lists... Done
 Building dependency tree
```



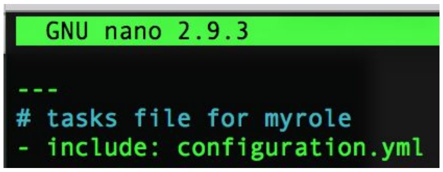
```
[ubuntu@master:/etc/ansible/roles$ tree myrole/
myrole/
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
ubuntu@master:/etc/ansible/roles$
```

- Now we are ready to create the tasks that our roles are supposed to perform
- Go inside the **tasks** folder -> Edit **main.yml**



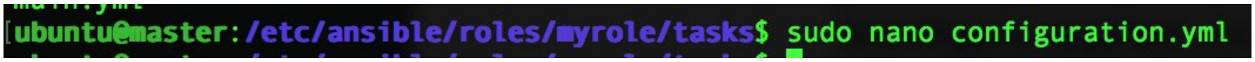
```
ubuntu@master:/etc/ansible/roles$ cd myrole/
ubuntu@master:/etc/ansible/roles/myrole$ ls
README.md  defaults  files  handlers  meta  tasks  templates  tests  vars
ubuntu@master:/etc/ansible/roles/myrole$ cd tasks
ubuntu@master:/etc/ansible/roles/myrole/tasks$ ls
main.yml
ubuntu@master:/etc/ansible/roles/myrole/tasks$ sudo nano main.yml
```

- Now we include the **configuration.yml** file from here

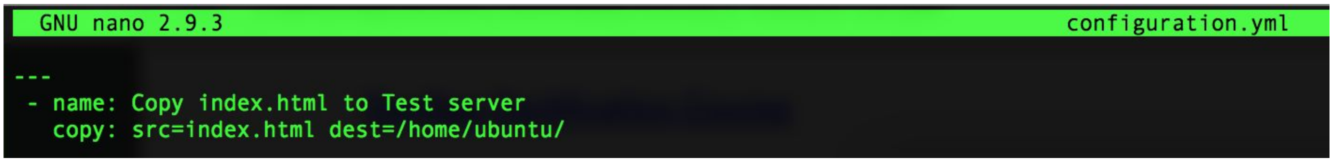


```
  GNU nano 2.9.3

---
# tasks file for myrole
- include: configuration.yml
```

- Now inside the **tasks** folder, we create our configuration.yml that we had before but modify a little bit

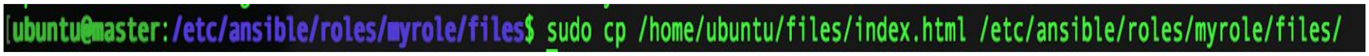


```
ubuntu@master:/etc/ansible/roles/myrole/tasks$ sudo nano configuration.yml
```



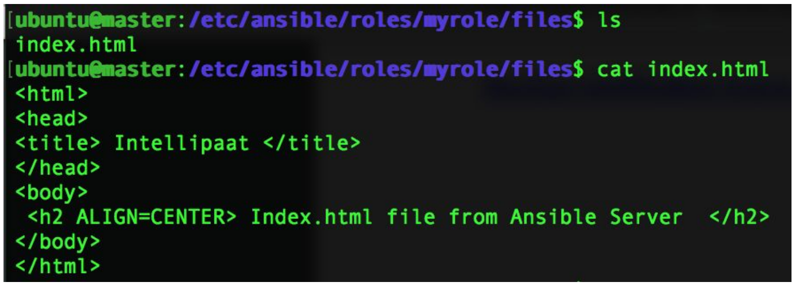
```
  GNU nano 2.9.3                                                        configuration.yml

---
 - name: Copy index.html to Test server
   copy: src=index.html dest=/home/ubuntu/
```

- Next go inside **files** and copy the index.html that we created earlier



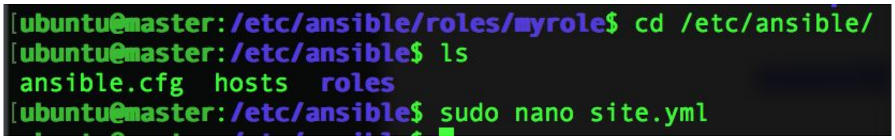
```
ubuntu@master:/etc/ansible/roles/myrole/files$ sudo cp /home/ubuntu/files/index.html /etc/ansible/roles/myrole/files/
```

- Make sure the file is there

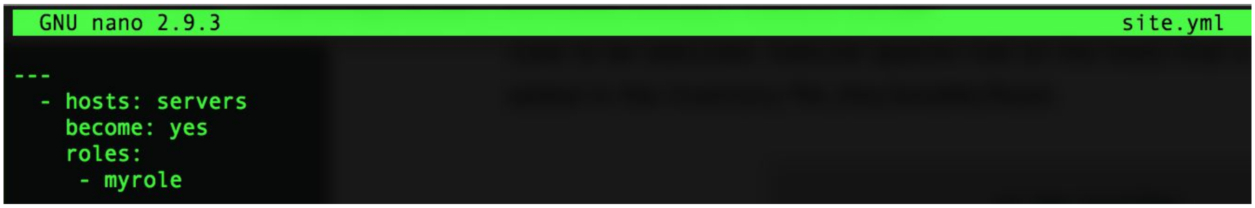


```
ubuntu@master:/etc/ansible/roles/myrole/files$ ls
index.html
ubuntu@master:/etc/ansible/roles/myrole/files$ cat index.html
<html>
<head>
<title> Intellipaat </title>
</head>
<body>
 <h2 ALIGN=CENTER> Index.html file from Ansible Server  </h2>
</body>
</html>
```

- No need for handlers in this case
- Go to **/etc/ansible/** and create one top level **.yml** file where we can add hosts and roles to be executed. Execute myrole role on the hosts that is under the group name servers, added in the inventory file **/etc/ansible/hosts**

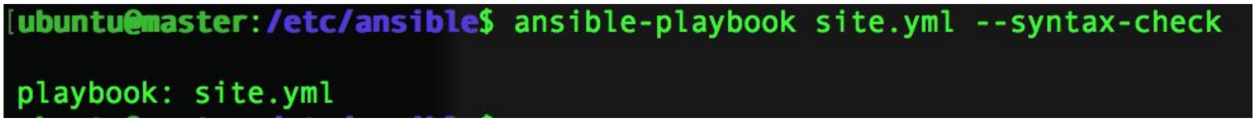


```
ubuntu@master:/etc/ansible/roles/myrole$ cd /etc/ansible/
ubuntu@master:/etc/ansible$ ls
ansible.cfg  hosts  roles
ubuntu@master:/etc/ansible$ sudo nano site.yml
```



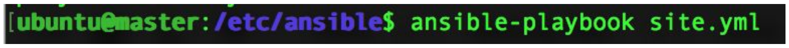
```
  GNU nano 2.9.3                                                         site.yml

---
  - hosts: servers
    become: yes
    roles:
     - myrole
```

- Before executing our top level **site.yml**, we check for syntax errors

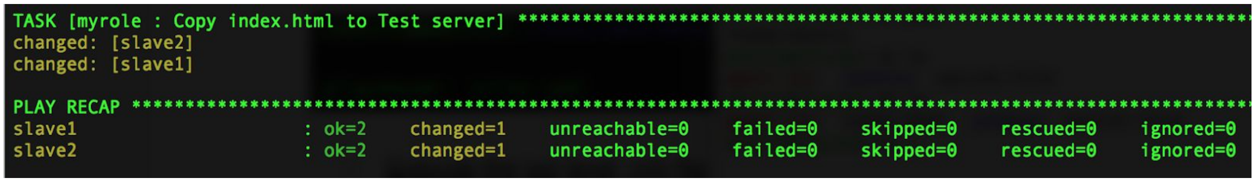


```
ubuntu@master:/etc/ansible$ ansible-playbook site.yml --syntax-check

playbook: site.yml
```

- Execute the top level .yml file

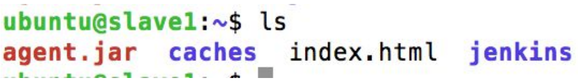


```
ubuntu@master:/etc/ansible$ ansible-playbook site.yml
```

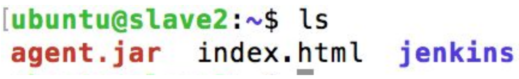


```
TASK [myrole : Copy index.html to Test server] ******************************************************
changed: [slave2]
changed: [slave1]

PLAY RECAP *****************************************************************************************
slave1                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
slave2                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

- Make sure the index.html is copied to the servers

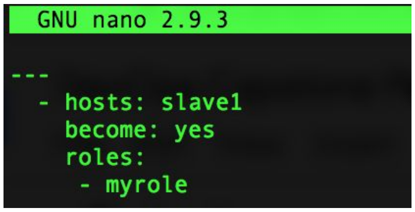


```
ubuntu@slave1:~$ ls
agent.jar  caches  index.html  jenkins
```



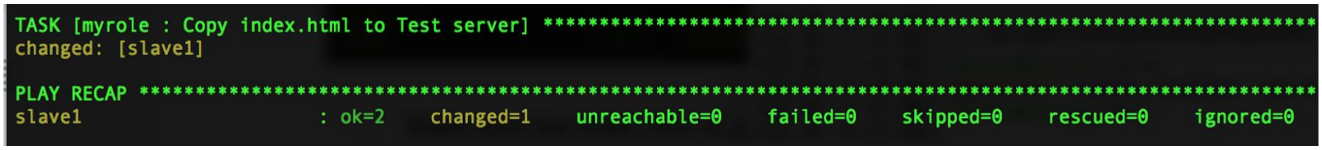
```
ubuntu@slave2:~$ ls
agent.jar  index.html  jenkins
```

- We'll modify the site.yml to only deploy to **slave1** since this is the Test server



```
  GNU nano 2.9.3

---
  - hosts: slave1
    become: yes
    roles:
     - myrole
```

- Now it should only deploy to slave1

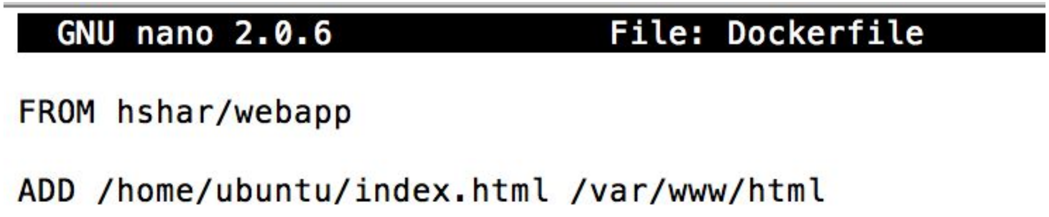


```
TASK [myrole : Copy index.html to Test server] ******************************************************
changed: [slave1]

PLAY RECAP *****************************************************************************************
slave1                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

- We've set up our task to be in an Ansible role

Step 5: Create a Dockerfile and use the **hshar/webapp** as a base image so there's already apache2 pre-installed once we run the container

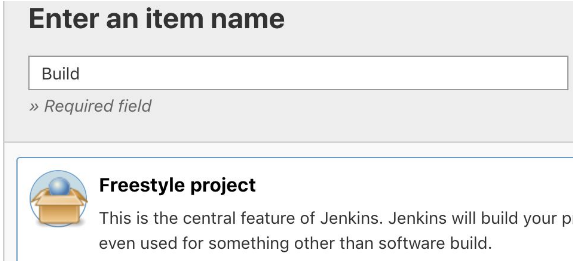
- Make sure to get the **index.html** file from **/home/ubuntu** of the **Test** server to **/var/www/html** of the container
- Push this to GItHub repository



```
GNU nano 2.0.6                    File: Dockerfile

FROM hshar/webapp

ADD /home/ubuntu/index.html /var/www/html
```

Step 6: Install Docker in both **Slave1** and **Slave2**

Step 7: Create **'Build'** Job in Jenkins for building our website



Enter an item name

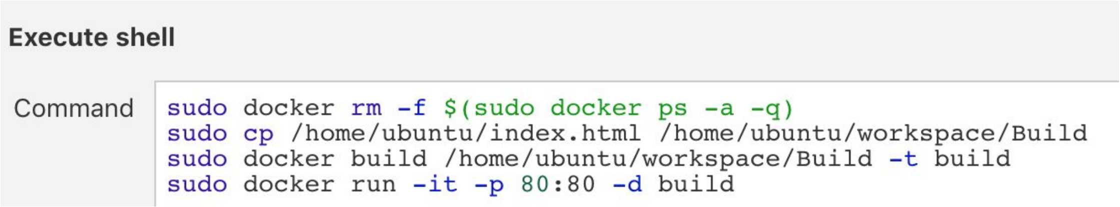
Build

» Required field

Freestyle project

This is the central feature of Jenkins. Jenkins will build your p
even used for something other than software build.

- Set the Github links to our Github repository
- Restrict project to run on **slave1**
- Branches to build: ***/hotfix**
- Check github hook trigger
- Build -> Execute shell -> Add commands

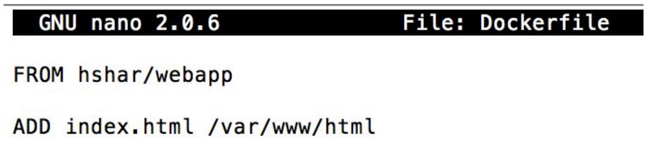


Execute shell

Command

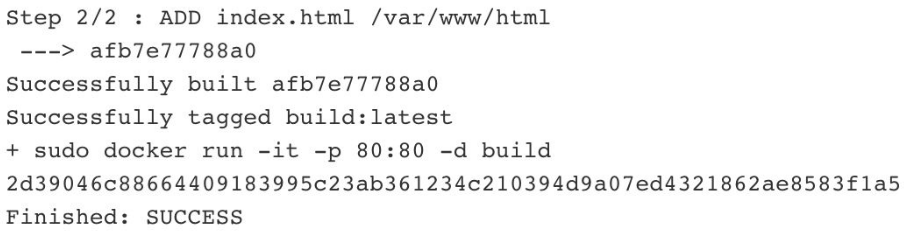
```
sudo docker rm -f $(sudo docker ps -a -q)
sudo cp /home/ubuntu/index.html /home/ubuntu/workspace/Build
sudo docker build /home/ubuntu/workspace/Build -t build
sudo docker run -it -p 80:80 -d build
```

- The line **sudo cp /home/ubuntu/index.html** is to copy the index.html from Ansible
- The reason for this is because the Dockerfile wasn't building when it comes to copying from **/home/ubuntu/index.html.** Instead, just copy the file from /home/ubuntu and transfer it to the same directory of the Dockerfile
- As a result, we have to modify the Dockerfile also, which is basically the original command that we had. The difference is this time, the file came from Ansible, not from our own repository



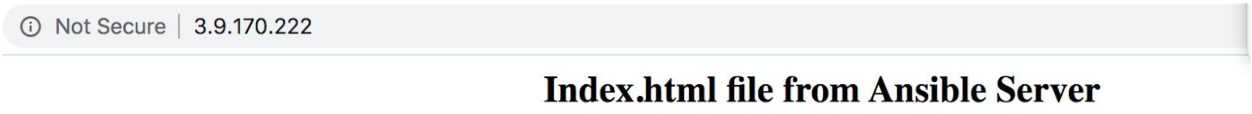
```
  GNU nano 2.0.6                 File: Dockerfile

FROM hshar/webapp

ADD index.html /var/www/html
```

- **Post-Build actions** -> **Build Other Projects** -> **Test** -> **Save**
- Make sure to run an arbitrary container in Test server
  sudo run -it -d ubuntu
- Build the job for now, see if it works



```
Step 2/2 : ADD index.html /var/www/html
 ---> afb7e77788a0
Successfully built afb7e77788a0
Successfully tagged build:latest
+ sudo docker run -it -p 80:80 -d build
2d39046c88664409183995c23ab361234c210394d9a07ed4321862ae8583f1a5
Finished: SUCCESS
```

- Check **<slave1-IP-Address>:80** to verify website is running

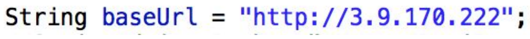


ⓘ Not Secure | 3.9.170.222

Index.html file from Ansible Server

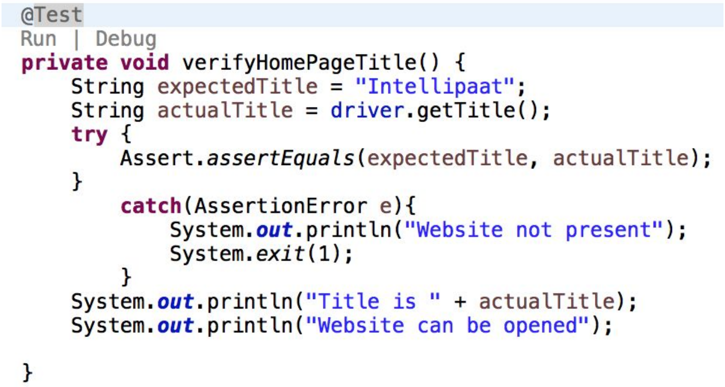
Step 8: Create the **Test** job for testing our website

- Design a test in Eclipse. The test will basically check if the title exists in the website. This should verify if the website is up and running
- Refer to DevOps Capstone Project 1 for designing and exporting the test case
- Make sure to change the **baseUrl** to the **slave1-IP-Address**



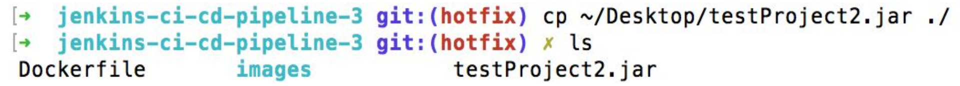
```
String baseUrl = "http://3.9.170.222";
```

- The test case that will verify if title is present or not

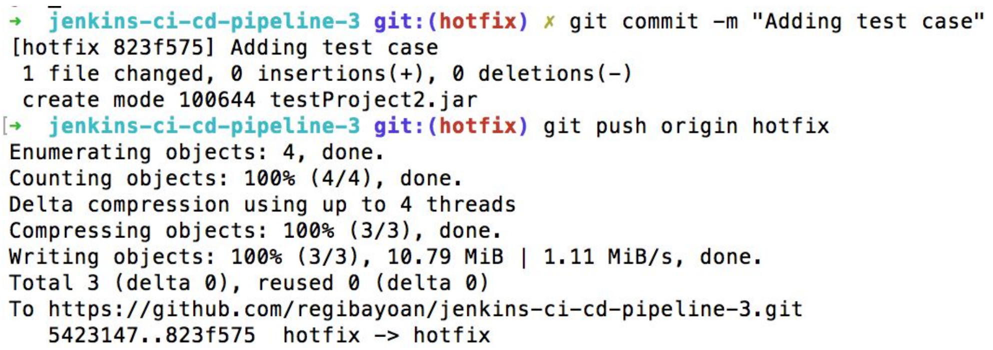


```
@Test
Run | Debug
private void verifyHomePageTitle() {
    String expectedTitle = "Intellipaat";
    String actualTitle = driver.getTitle();
    try {
        Assert.assertEquals(expectedTitle, actualTitle);
    }
        catch(AssertionError e){
            System.out.println("Website not present");
            System.exit(1);
        }
    System.out.println("Title is " + actualTitle);
    System.out.println("Website can be opened");

}
```

- Push the runnable jar file to Github as well so the **Test** job can just pick it up from there



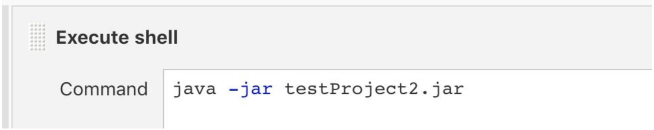
```
[→  jenkins-ci-cd-pipeline-3 git:(hotfix) cp ~/Desktop/testProject2.jar ./
[→  jenkins-ci-cd-pipeline-3 git:(hotfix) ✗ ls
 Dockerfile       images          testProject2.jar
```



```
→  jenkins-ci-cd-pipeline-3 git:(hotfix) ✗ git commit -m "Adding test case"
[hotfix 823f575] Adding test case
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 testProject2.jar
[→  jenkins-ci-cd-pipeline-3 git:(hotfix) git push origin hotfix
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 10.79 MiB | 1.11 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/regibayoan/jenkins-ci-cd-pipeline-3.git
   5423147..823f575  hotfix -> hotfix
```

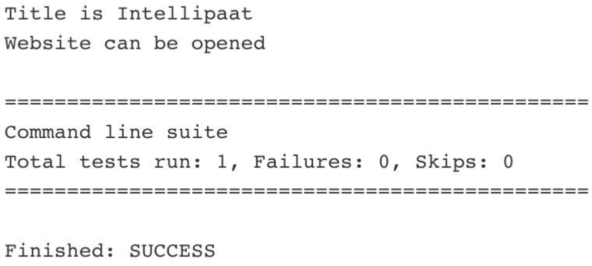
- Install **chromedriver** in slave1
- Install **Google Chrome** in slave1

- Now let's add the commands in the **Test** job



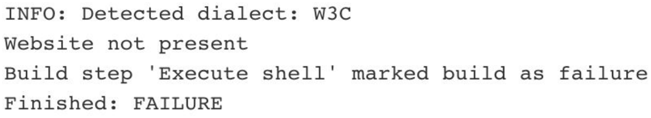
Command
```
java -jar testProject2.jar
```

- Also make sure to link the Github links
- Restrict project to run on **slave1**
- Branch to build -> ***/hotfix**
- Check Github hook trigger
- Add **Post-Build Actions** -> **PushToDockerHub**
- Try running the **Test** job and see if it works



```
Title is Intellipaat
Website can be opened

===============================================
Command line suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================

Finished: SUCCESS
```

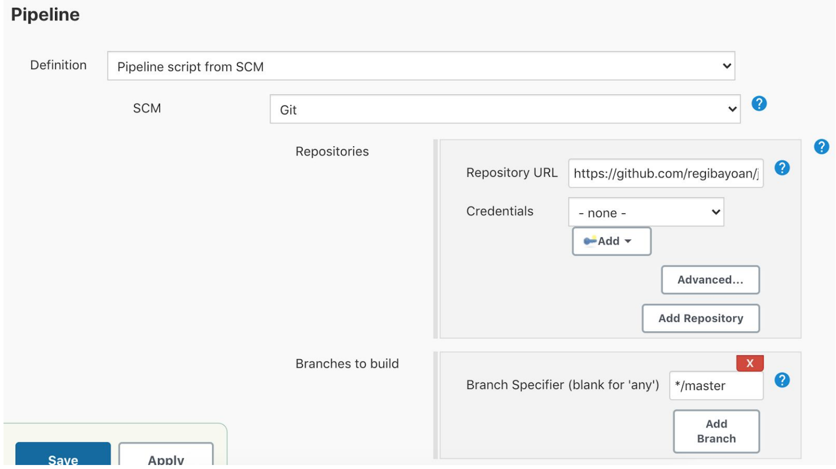
- To verify, try stopping the container to see if the job fails



```
INFO: Detected dialect: W3C
Website not present
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Step 9: Create the **PushToDockerHub** job for pushing Docker image to Docker Hub

- Install **Docker Pipeline** plugin in Jenkins
- Create a new job -> **PushToDockerHub** -> **Pipeline**



Pipeline

Definition: Pipeline script from SCM

SCM: Git

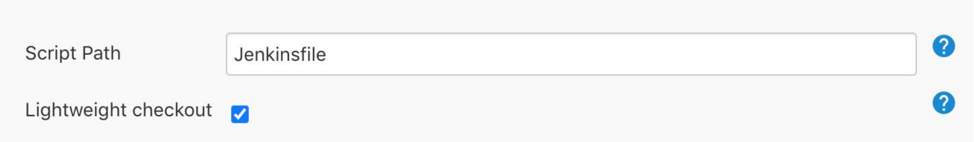
Repositories

Repository URL: https://github.com/regibayoan/

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master



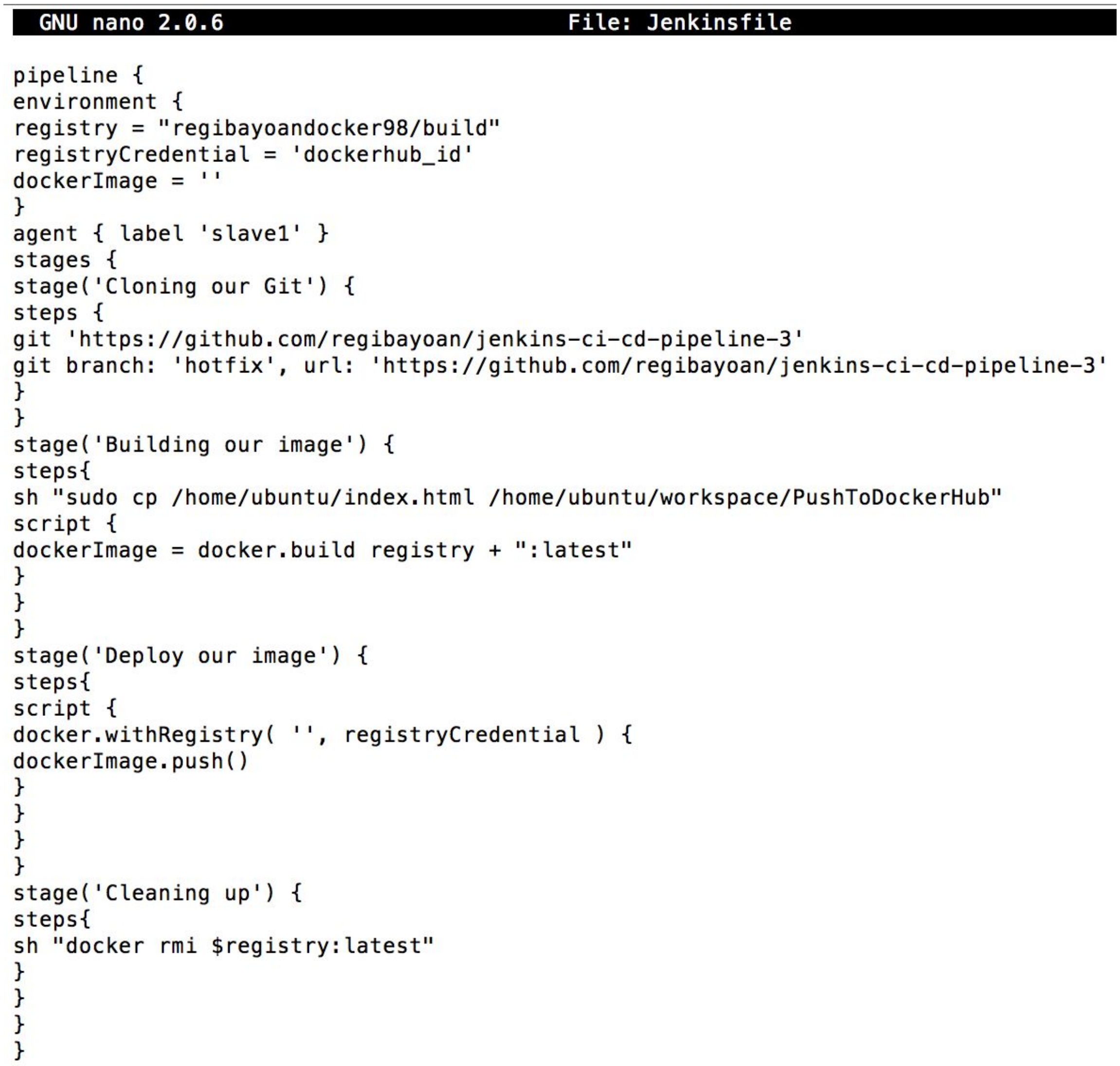
Script Path: Jenkinsfile

Lightweight checkout ☑

- Create a Jenkinsfile and push to our GitHub repository. Jenkins will look for Jenkinsfile when running this job
- Set links to our repository
- **Build Triggers** -> **Build after projects are built** -> **Test** -> **Trigger only if build is stable**

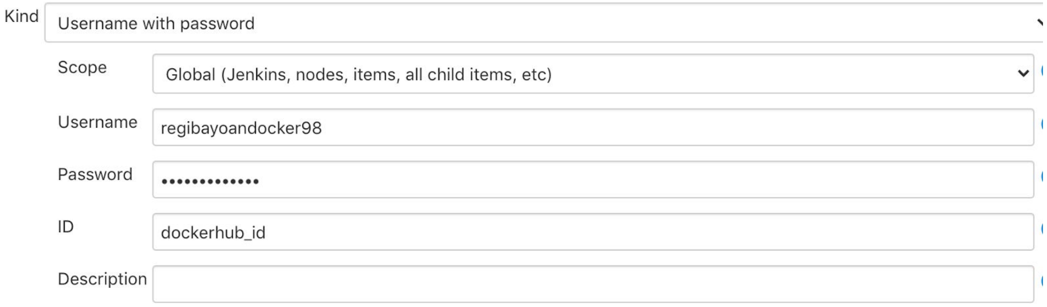
  Jenkinsfile code:

  - Set **registry** = **<registry_name>**
  - registryCredential = **<setup in Docker credentials>**
  - agent { label 'slave1' } = tells Jenkins to restrict job to only **slave1**
  - **git branch** command just after the cloning the repository is needed to change the branch to **hotfix**, otherwise it will default to **master** branch
  - For **Building our image** step, added the **sh command** to copy the index.html file to the **PushToDockerHub workspace** so that it will be in the same directory as the **Dockerfile.** The Dockerfile will build the index.html file within its directory. If this step is not added, there will be a file not found error because the newly created workspace doesn't have the index.html file, which contents come from the hotfix branch repository from Github



```
GNU nano 2.0.6                          File: Jenkinsfile

pipeline {
environment {
registry = "regibayoandocker98/build"
registryCredential = 'dockerhub_id'
dockerImage = ''
}
agent { label 'slave1' }
stages {
stage('Cloning our Git') {
steps {
git 'https://github.com/regibayoan/jenkins-ci-cd-pipeline-3'
git branch: 'hotfix', url: 'https://github.com/regibayoan/jenkins-ci-cd-pipeline-3'
}
}
stage('Building our image') {
steps{
sh "sudo cp /home/ubuntu/index.html /home/ubuntu/workspace/PushToDockerHub"
script {
dockerImage = docker.build registry + ":latest"
}
}
}
stage('Deploy our image') {
steps{
script {
docker.withRegistry( '', registryCredential ) {
dockerImage.push()
}
}
}
}
stage('Cleaning up') {
steps{
sh "docker rmi $registry:latest"
}
}
}
}
```

- **Setup Docker credentials**
    - **Manage Jenkins** -> **Manage Credentials** -> **System** -> **Global Credentials (Unrestricted)** -> **Add Credentials**



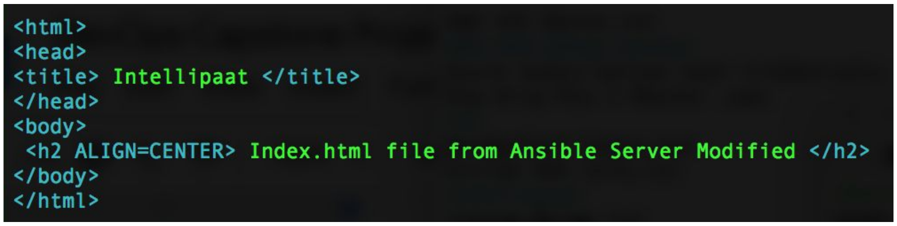
Kind	Username with password
Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	regibayoandocker98
Password	•••••••••••••
ID	dockerhub_id
Description	

- Add Jenkins slave user to Docker group to avoid **“permission denied”** error
- **sudo group add docker** -> **sudo usermod -aG docker $USER**

- If still getting an error change permission of the **/var/run/docker.sock**
    - **chmod 777 /var/run/docker.sock**

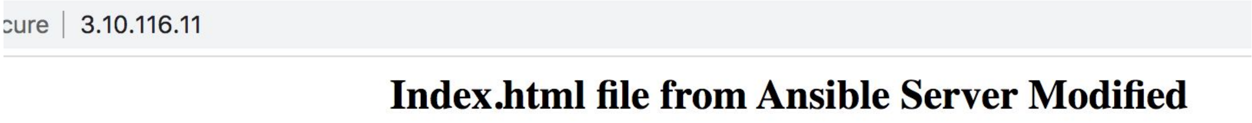
- Run the **PushToDockerHub** job to verify if image is pushed to Docker Hub, it should be in Docker Hub

- Lastly, to verify our pipeline so far. Let's modify the **index.html** from our **Ansible role** and deploy it to **slave1**



```
<html>
<head>
<title> Intellipaat </title>
</head>
<body>
 <h2 ALIGN=CENTER> Index.html file from Ansible Server Modified </h2>
</body>
</html>
```

- Run the **Build** Job. If successful, the newly pushed image should reflect on Docker Hub
- Pull image from Github and run container from **slave2**

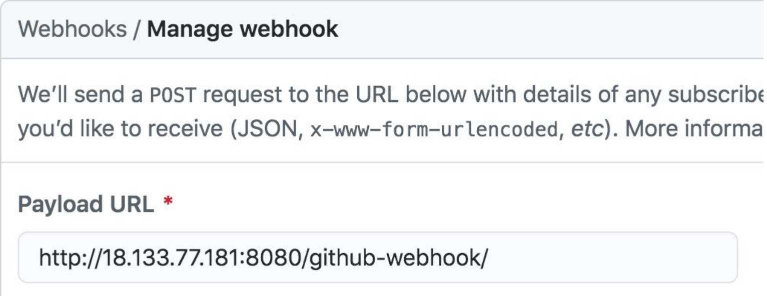


cure | 3.10.116.11

Index.html file from Ansible Server Modified

Step 10: Now we configure **GitHub Webhook** so the pipeline will automatically build the application when there is a commit to **hotfix** branch

- Go to Jenkins Dashboard -> **Build** -> **Configure** -> **Build Triggers** -> Check the **GitHub hook trigger for GITScm polling** -> **Save**
- Next in **GitHub -> Settings ->** Click on **Webhooks -> Add Webhooks ->** Insert Jenkins server address as shown **JenkinsServerIPAdress:8080/github-webhook/**



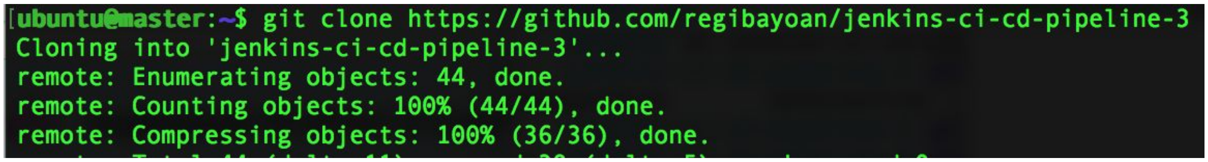
Webhooks / **Manage webhook**

We'll send a POST request to the URL below with details of any subscribe
you'd like to receive (JSON, x-www-form-urlencoded, *etc*). More informa

Payload URL *

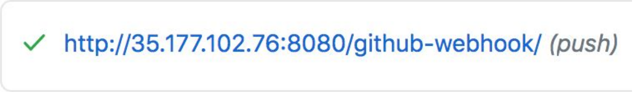
http://18.133.77.181:8080/github-webhook/

- Go to the master terminal to trigger a build



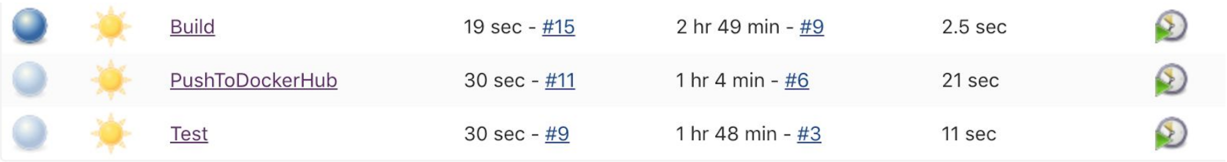
```
ubuntu@master:~$ git clone https://github.com/regibayoan/jenkins-ci-cd-pipeline-3
Cloning into 'jenkins-ci-cd-pipeline-3'...
remote: Enumerating objects: 44, done.
remote: Counting objects: 100% (44/44), done.
remote: Compressing objects: 100% (36/36), done.
```

- We should see a check mark on the webhook



✓ http://35.177.102.76:8080/github-webhook/ *(push)*

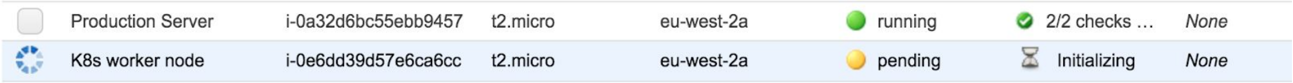
- Verify by making a commit to **hotfix** branch
- The build should start automatically like below



●	☀	Build	19 sec - #15	2 hr 49 min - #9	2.5 sec
●	☀	PushToDockerHub	30 sec - #11	1 hr 4 min - #6	21 sec
●	☀	Test	30 sec - #9	1 hr 48 min - #3	11 sec

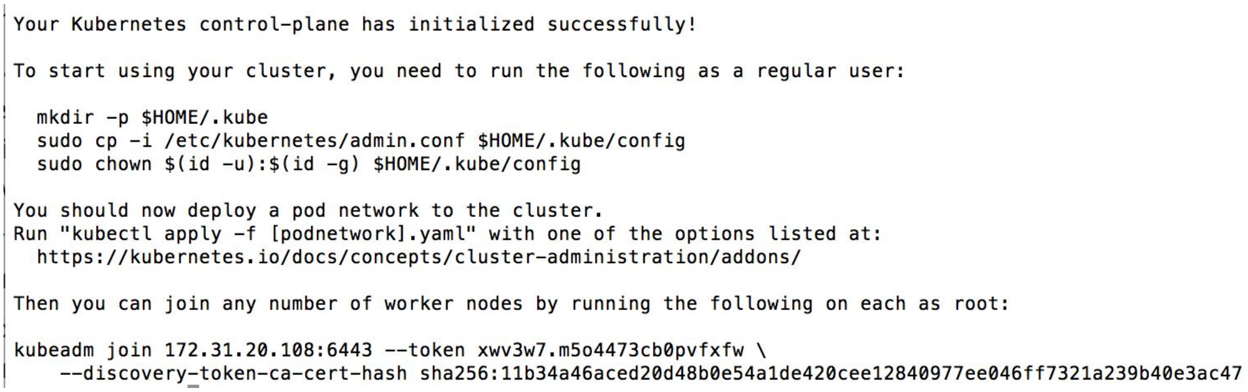
Step 11: Before creating the **PushToProduction** job. First, set up a Kubernetes cluster

- Create a 4th EC2 instance, which is the **k8s worker node**
- We will set up **slave2** (production server) to be the **K8s master node.** This EC2 instance has to have at least 2 vCPUs to work properly.



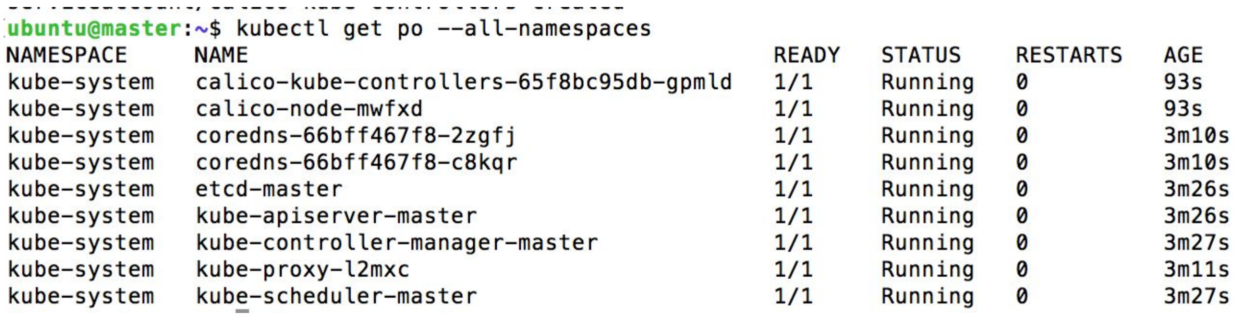
□	Production Server	i-0a32d6bc55ebb9457	t2.micro	eu-west-2a	● running	✓ 2/2 checks ...	*None*
	K8s worker node	i-0e6dd39d57e6ca6cc	t2.micro	eu-west-2a	● pending	⌛ Initializing	*None*

- Install Kubernetes using kubeadm method
- At the end of Kubernetes installation, we should output like below



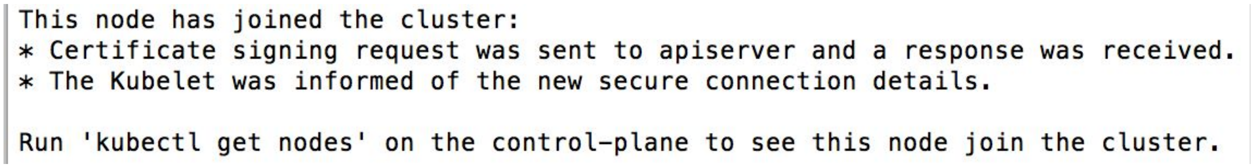
```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.20.108:6443 --token xwv3w7.m5o4473cb0pvfxfw \
    --discovery-token-ca-cert-hash sha256:11b34a46aced20d48b0e54a1de420cee12840977ee046ff7321a239b40e3ac47
```

- Add the Calico network



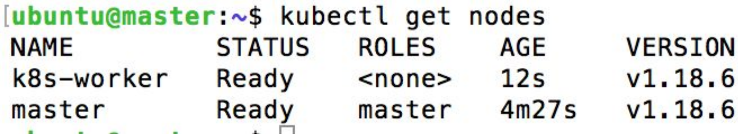
```
ubuntu@master:~$ kubectl get po --all-namespaces
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-65f8bc95db-gpmld   1/1     Running   0          93s
kube-system   calico-node-mwfxd                          1/1     Running   0          93s
kube-system   coredns-66bff467f8-2zgfj                   1/1     Running   0          3m10s
kube-system   coredns-66bff467f8-c8kqr                   1/1     Running   0          3m10s
kube-system   etcd-master                                1/1     Running   0          3m26s
kube-system   kube-apiserver-master                      1/1     Running   0          3m26s
kube-system   kube-controller-manager-master             1/1     Running   0          3m27s
kube-system   kube-proxy-l2mxc                           1/1     Running   0          3m11s
kube-system   kube-scheduler-master                      1/1     Running   0          3m27s
```

- Add the k8s worker node to the cluster



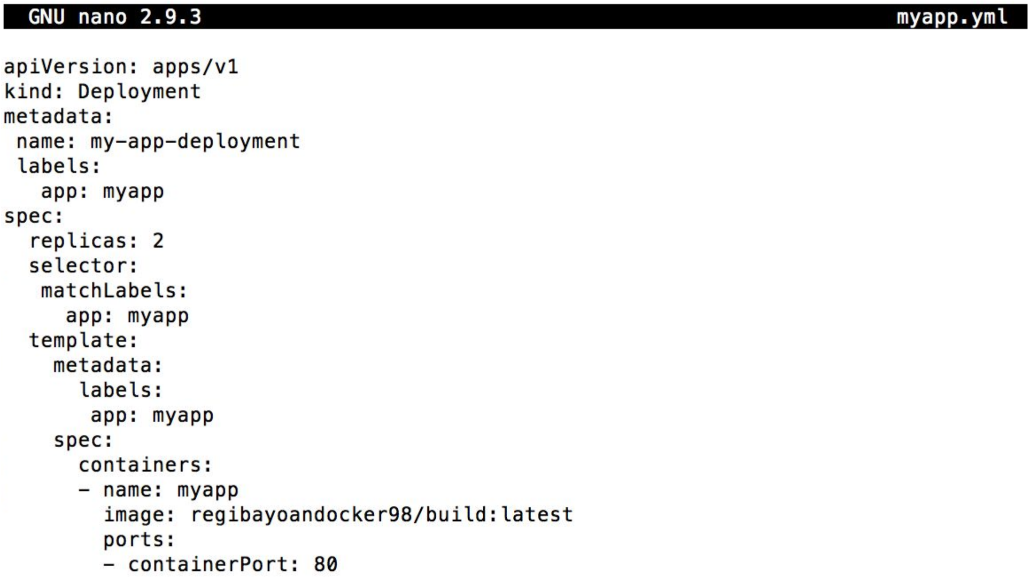
```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

- Check if the worker node has joined the cluster



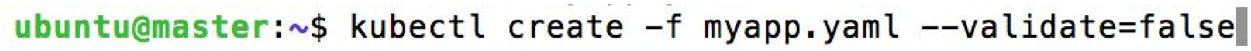
```
ubuntu@master:~$ kubectl get nodes
NAME         STATUS   ROLES    AGE     VERSION
k8s-worker   Ready    <none>   12s     v1.18.6
master       Ready    master   4m27s   v1.18.6
```

- Create the deployment yaml file for our website
- Set **replicas to 2**
- Set image as **regibayoandocker98/build:latest** since this is the image we want to pull from Docker Hub then deploy from our Kubernetes cluster



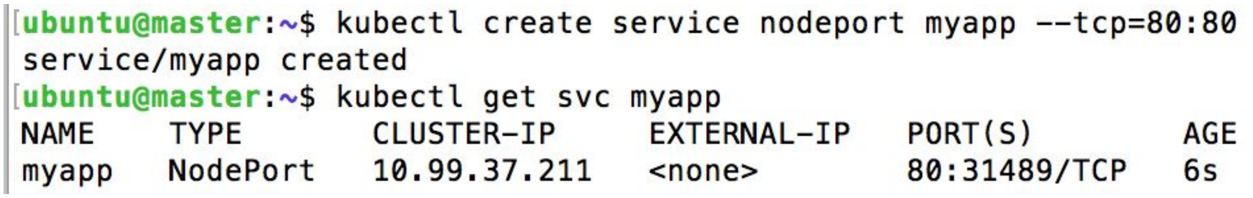
```
 GNU nano 2.9.3                                                     myapp.yml

apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-app-deployment
 labels:
   app: myapp
spec:
  replicas: 2
  selector:
   matchLabels:
     app: myapp
  template:
    metadata:
      labels:
       app: myapp
    spec:
      containers:
      - name: myapp
        image: regibayoandocker98/build:latest
        ports:
        - containerPort: 80
```

- Apply the deployment



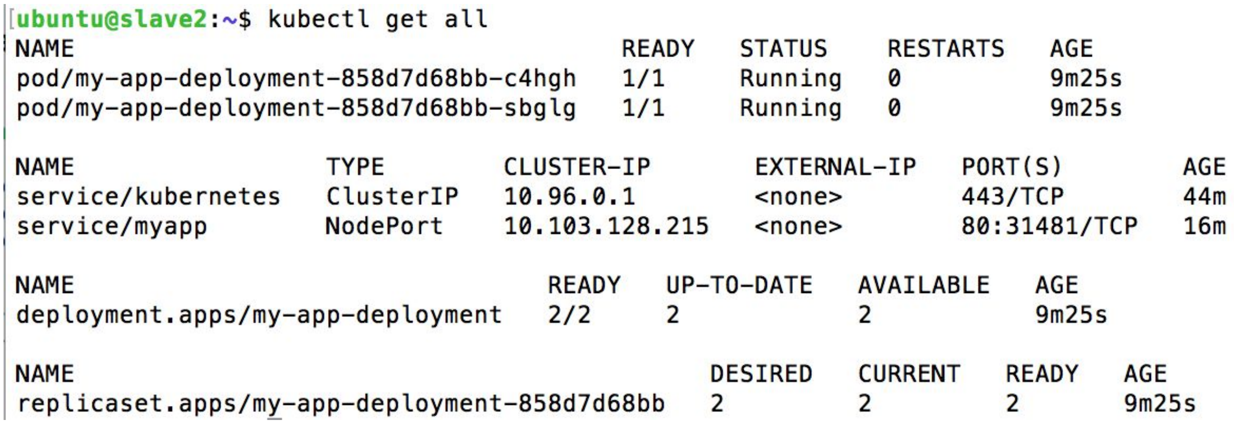
```
ubuntu@master:~$ kubectl create -f myapp.yaml --validate=false
```

- Create a service to expose our application and verify the port



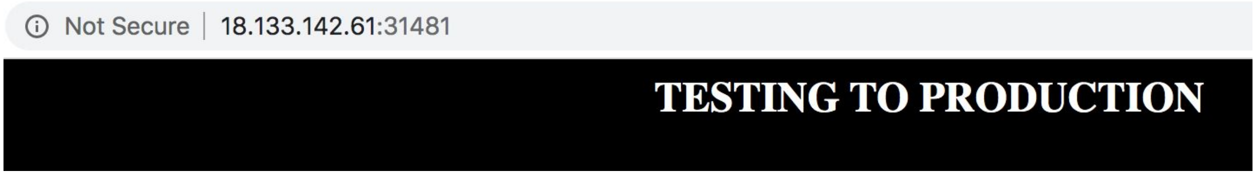
```
[ubuntu@master:~$ kubectl create service nodeport myapp --tcp=80:80
service/myapp created
[ubuntu@master:~$ kubectl get svc myapp
NAME     TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
myapp    NodePort   10.99.37.211   <none>        80:31489/TCP   6s
```

- Get the status of the Kubernetes components, Make sure the pods have the right amount of replicas



```
[ubuntu@slave2:~$ kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/my-app-deployment-858d7d68bb-c4hgh   1/1     Running   0          9m25s
pod/my-app-deployment-858d7d68bb-sbglg   1/1     Running   0          9m25s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        44m
service/myapp        NodePort    10.103.128.215   <none>        80:31481/TCP   16m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-app-deployment   2/2     2            2           9m25s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/my-app-deployment-858d7d68bb   2         2         2       9m25s
```

- Visit the webpage. The website can be launched either on the master node or worker node, then point to the port provided by the service

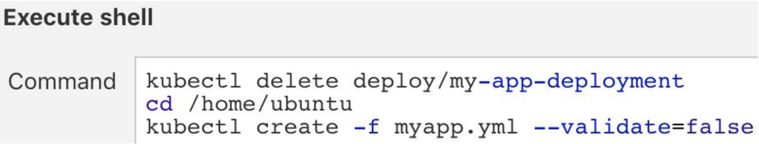


ⓘ Not Secure | 18.133.142.61:31481

TESTING TO PRODUCTION

Step 12: Integrate Kubernetes with Jenkins by creating the **PushToProduction** job

- Set the github repo link
- Restrict to run at **slave2** since this is now in **Production** server
- Branch to build ***/hotfix** branch
- Build Triggers -> Build after projects are built -> PushToDockerHub
- Build -> Execute shell -> Add commands

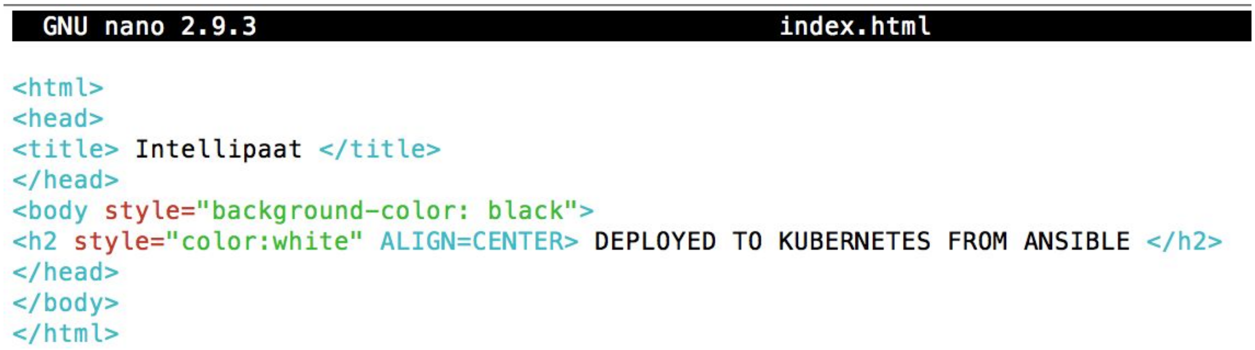


Execute shell

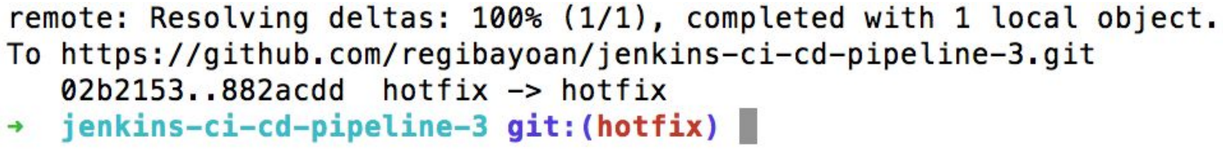
Command

```
kubectl delete deploy/my-app-deployment
cd /home/ubuntu
kubectl create -f myapp.yml --validate=false
```

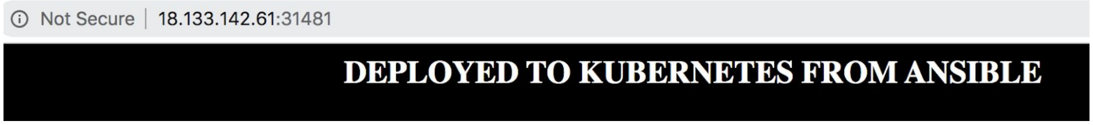
- For this to work, we have to run one deployment first for it to be deleted, just like running one arbitrary container in Docker. Leave the service that was already created earlier
- Now we just create a new deployment. The service will automatically attach to this deployment and expose our application to the internet.
- To verify this is working, change the index.html from Ansible -> Run the ansible playbook to transfer file to Test server -> Commit to hotfix branch -> The build should start and reflect all the way to the Kubernetes cluster -> Verify website is displayed in both Production server node and k8s-worker node



```
  GNU nano 2.9.3                                    index.html

<html>
<head>
<title> Intellipaat </title>
</head>
<body style="background-color: black">
<h2 style="color:white" ALIGN=CENTER> DEPLOYED TO KUBERNETES FROM ANSIBLE </h2>
</head>
</body>
</html>
```

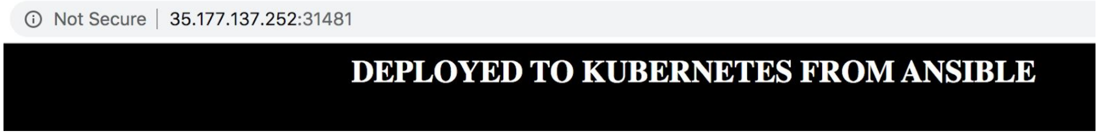


```
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/regibayoan/jenkins-ci-cd-pipeline-3.git
   02b2153..882acdd  hotfix -> hotfix
➜  jenkins-ci-cd-pipeline-3 git:(hotfix)
```



ⓘ Not Secure | 18.133.142.61:31481

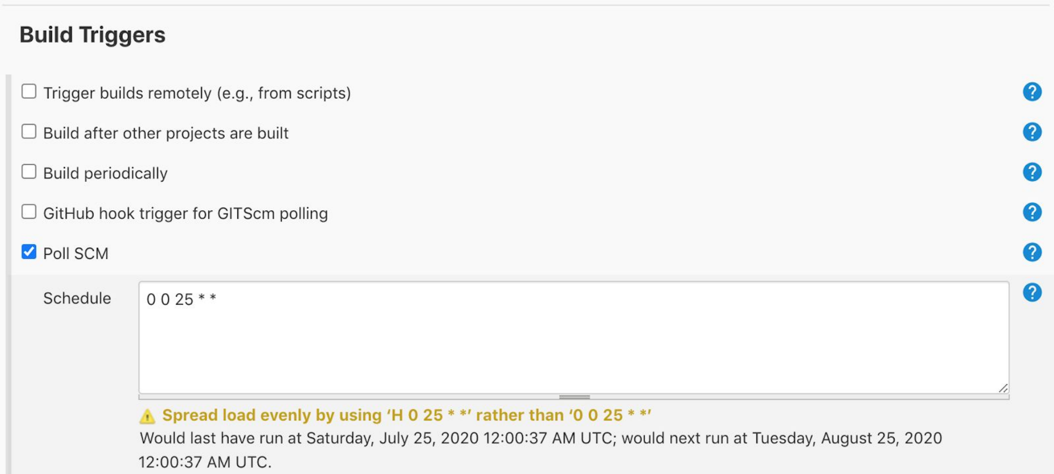
DEPLOYED TO KUBERNETES FROM ANSIBLE



ⓘ Not Secure | 35.177.137.252:31481

DEPLOYED TO KUBERNETES FROM ANSIBLE

Step 13: Lastly, we create a new pipeline for the **master** branch. Pretty much the same pipeline but modify the Build stage to build from the **master** branch. Also, set the **Build Trigger**s to be triggered every 25th of the month using **Poll SCM**.



Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Build periodically

☐ GitHub hook trigger for GITScm polling

☑ Poll SCM

Schedule: 0 0 25 * *

⚠ **Spread load evenly by using 'H 0 25 * *' rather than '0 0 25 * *'**

Would last have run at Saturday, July 25, 2020 12:00:37 AM UTC; would next run at Tuesday, August 25, 2020 12:00:37 AM UTC.